The Effects of Over and Under Sampling on Fault-prone Module Detection

Yasutaka Kamei, Akito Monden, Shinsuke Matsumoto,
Takeshi Kakimoto, Ken-ichi Matsumoto
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
+81 743 72 5312
{yasuta-k, akito-m, shinsuke-m, takesi-k, matumoto}@is.naist.jp

Abstract

The goal of this paper is to improve the prediction performance of fault-prone module prediction models (fault-proneness models) by employing over/under sampling methods, which are preprocessing procedures for a fit dataset. The sampling methods are expected to improve prediction performance when the fit dataset is imbalanced, i.e. there exists a large difference between the number of fault-prone modules and not-fault-prone modules. So far, there has been no research reporting the effects of applying sampling methods to fault-proneness models. In this paper, we experimentally evaluated the effects of four sampling methods (random over sampling, synthetic minority over sampling, random under sampling and one-sided selection) applied to four fault-proneness models (linear discriminant analysis, logistic regression analysis, neural network and classification tree) by using two module sets of industry legacy software. All four sampling methods improved the prediction performance of the linear and logistic models, while neural network and classification tree models did not benefit from the sampling methods. The improvements of F1-values in linear and logistic models were 0.078 at minimum, 0.224 at maximum and 0.121 at the mean.

1. Introduction

Identification of fault-prone modules [12] is an important issue for improving software quality in the testing and maintenance phases [9]. Various multivariate modeling techniques, which are applicable to fault-prone module prediction, have been proposed, including linear discriminant analysis [12][13], logistic regression analysis [10], neural network [2] and classification tree [5]. These models are built from a fit dataset, which contains product metrics and fault data of modules of a past software project.

However, preparing a *balanced* fit dataset is not always possible in practical situations. Many module datasets in the field are actually *imbalanced*, i.e. there exists a large difference between the number of fault-prone modules and not-fault-prone modules; and, this causes performance degradation of fault-proneness models [6]. For example, in Khoshgoftaar et al.'s case [7], the percentage of fault-prone modules to all modules (P_{fp}) was about 6% in the maintenance phase. Also, in NASA IV&V Facility Metrics Data Program's case [11], P_{fp} was below 15% in 8 datasets out of 15.

So far, an extended classification tree model suitable for an imbalanced dataset has been proposed by Khoshgoftaar et al [7]. In conventional fault-prone models, the prediction accuracy of the minority class (fault-prone modules) usually becomes worse since the prediction accuracy of the majority class (not-fault prone modules) is dominant in satisfying objective functions of the models [1]. The extended classification tree model is built based on the misclassification rate to improve the accuracy of the minority class. However, the problem here is that the classification tree is not always the best (most accurate) model among other models. Gray and Macdonell showed that the best model among available models often depends on a dataset being used [2].

In this paper, we focus on an approach to modify a fit dataset and not to extend fault-proneness models themselves. The approach is called sampling methods that can be applied independent of a fault-prone model. The sampling methods are classified into two methods: (1) *over sampling* that adds fake fault-prone modules (minority class) to a fit dataset, and (2) *under sampling* that reduces not-fault-prone modules

0-7695-2886-4/07 $20.00 © 2007 IEEE
DOI 10.1109/ESEM.2007.28

(majority class) from the fit dataset [1][8]. However, to our knowledge, no study has reported the effects of applying sampling methods to fault-proneness models. Furthermore, it is not clear which sampling methods are most appropriate for the fault-proneness models.

In this paper, we experimentally evaluated the effects of four sampling methods (random over sampling, synthetic minority over sampling, random under sampling and one-sided selection) applied to four commonly used fault-proneness models (linear discriminant analysis, logistic regression analysis, neural network and classification tree) using two module datasets A and B (each written with different programming language) of large legacy software of a Japanese software company. Since all four sampling methods have the ability to control the number of modules to add to or delete from a dataset, we also evaluated the prediction accuracy with different numbers of additions/deletions. In our experiment, the fit datasets (of both A and B module sets) were built from data during the three years prior to a certain release in a maintenance phase (P_{fp} are 5.67% and 13.16%), and the test datasets were built from data during three years after the release (P_{fp} are 2.15% and 3.36%). The predictor variables of fault-proness models are 16 complexity metrics and 3 change history metrics of modules.

In what follows, Section 2 describes the details of the sampling methods. Section 3 provides the design of our experiment, and Section 4 gives the results and discussion. Section 5 summarizes the paper and presents some future topics of research.

2. Sampling Methods

In this paper, “sampling” means a preprocessing procedure to correct the imbalance of a given dataset by increasing or decreasing the cases (modules) in the dataset before applying it to model building. There are two types of sampling: over sampling and under sampling. Over sampling increases minority cases, while under sampling decreases majority cases of the dataset.

Generally, the prediction accuracy of the minority class (fault-prone modules) becomes worse since the prediction accuracy of the majority class (not-fault prone modules) is dominant in satisfying objective functions of the models [1]. By correcting the imbalance, the prediction accuracy of the minority case is expected to be improved. One of the concerns in using sampling methods is that over sampling might cause over fitting to fake (added) cases. Another concern is that under sampling might eliminate useful cases. Therefore, experimental evaluation of sampling methods is needed.

In this paper, we use the following sampling methods in the experiment.

2.1. Over Sampling

2.1.1. Random over sampling (ROS)

Random over sampling (ROS) increases the number of minority cases in a dataset by duplicating minority cases randomly. ROS repeats until P_{fp} reaches a predefined value (e.g. 50%) as follows.

Step 1: Selection of a minority case

One case is selected randomly from a minority class in a dataset.

Step 2: Duplication of a minority case

A new case is added to the dataset by duplicating the case selected in step 1.

2.1.2. Synthetic Minority Over sampling Technique (SMOTE)

Chawla et al. proposed Synthetic Minority Over sampling Technique (SMOTE) that produces new cases based on the k-nearest neighbor [1]. SMOTE repeats steps the following steps 1,…,4 for each case of minority class.

Step 1: Selection of a minority case

One case (denoted as m_a) is selected from a minority class in a dataset.

Step 2: Identification of k-nearest neighbors

K-nearest neighbors of m_a are identified based on the similarity computation. In this paper, we adopted $k=5$ as Chawla et al. used [1]. Below describes how to compute the similarity.

Step 2-1. The normalization of predictor variables
SMOTE requires computing the similarity between two arbitrary cases in a dataset to identify the k-nearest neighbors of a case. However, the similarity calculation based on predictor variables of software modules requires some normalization, because the value ranges of the variables widely vary. Hence, we added a new step, which normalizes predictor variables so that their value ranges become [0, 1]. The normalized value $norm(v_{i,j})$ of variable f_j of case m_i is calculated by the following equation:

$$norm(v_{i,j}) = \frac{v_{i,j} - min(f_j)}{max(f_j) - min(f_j)} \;,$$

where $min(f_j)$ and $max(f_j)$ denote maximum and minimum value in variable f_j, respectively.

Step 2-2: Similarity calculation

The similarity between m_a and all other cases in the minority class is calculated. In this paper, we used Euclidian distance, which is widely used as similarity measure. The similarity $sim(m_a,m_i)$ between target case m_a and other case m_i is calculated by the following equation:

$$sim(m_a, m_i) = \sqrt{\sum_{j=1}^{n} (v_{a,j} - v_{i,j})^2} \;,$$

where $v_{i,j}$ denotes a value of variable f_j of case m_i, and n denotes the number of predictor variables.

Step 3: Selection of a neighbor

One case m_r was selected randomly from the k-nearest neighbors.

Step 4: Addition of a minority case

A new case was added in some places along a straight line between the vertexes of the feature vector of cases m_a and m_r. Step 4 was repeated x times, where x = (the number of additional minority cases) / (the number of original minority cases).

2.2. Under Sampling

2.2.1. Random under sampling (RUS)

Random under sampling (RUS) decreases the number of majority cases in a dataset by deleting majority cases randomly. RUS repeats until P_{fp} reaches a predefined value as follows.

Step 1: Selection of a majority case

One case is selected randomly from a majority class in a dataset.

Step 2: Deletion of a case

The case selected in step 1 is deleted from the dataset.

2.2.2. One-sided selection (ONESS)

Kubat et al. [8] proposed an under sampling method called one-sided selection (ONESS), which exploits the concept of Tomek links [15]. Denoted by $\delta(x,y)$, the distance between x and y, a pair of cases (x,y) is called a Tomek link if no case z exists such that $\delta(x,z) < \delta(x,y)$ or $\delta(y,z) < \delta(y,x)$ [15]. Kubat et al. proposed to select x from a minority class and y from a majority class. In this case, a Tomek link (x,y) can be found either (1) on the class boundary when both x and y exist in the right class regions, or (2) inside one of the

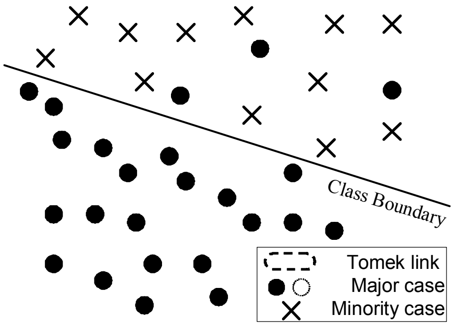

Figure 1. Example of distribution of majority cases and minority cases (Original fit dataset S)

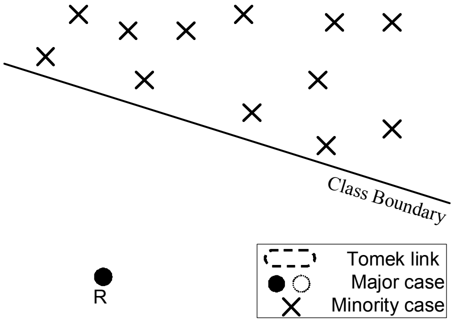

Figure 2. Set C of cases containing R and all minority cases (After step 1)

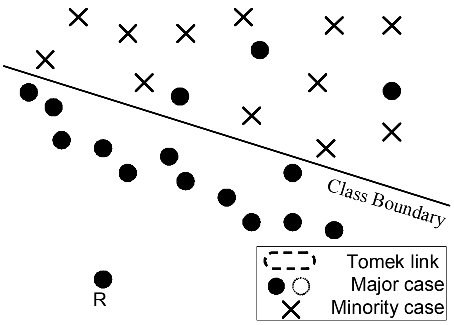

Figure 3. Set C without redundant (majority) cases (After step 2)

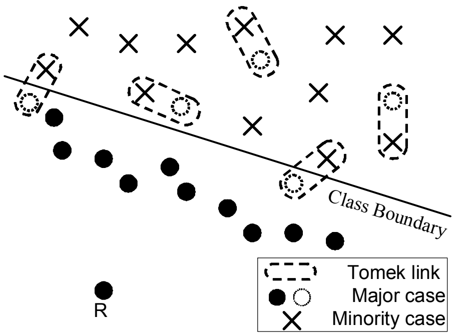

Figure 4. Tomek links in set C

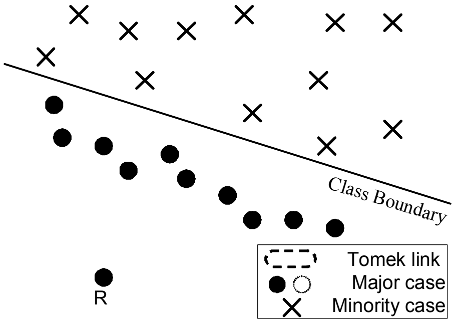

Figure 5. Resulting set T (After step 4)

class regions when either x or y exists in the wrong region (i.e. it is considered noise). Kubat proposed to delete a majority case in a Tomek link that is believed to be borderline and/or noisy.

Futhermore, Kubat et al. proposed to remove redundant (majority) cases based on a 1-nearest neighbor computation.

Below are the details of one-side selection.

Step 1: Selection of a majority case and initial samples

Let *S* be the original fit dataset (Fig. 1). Randomly select a case *R* from a majority class in *S*. Let *C* be a set of cases containing *R* and all minority cases (Fig. 2).

Step 2: Building a sample set having no redundant majority cases

Classify *S* with the 1-nearest neighbor rule using cases in *C*, and compare the assigned classification labels (either fault-prone or not-fault prone) with the original ones. Move all misclassified cases into *C*, which is now consistent with *S* while being smaller. As a result, redundant (majority) cases do not exist in *C* (Fig. 3).

Step 3: Deletion of cases in Tomek links

Remove from *C* all majority cases participating in Tomek links (Fig. 4). This removes those majority cases that are believed to be borderline and/or noisy. All positive examples are retained. The resulting set is referred to as *T* (Fig. 5).

Step 4: Repetition

In our pilot experiment, it turned out that the number of majority cases eliminated depends strongly on *R* selected in Step 1 (note that *R* is randomly selected). Thus, this paper proposes to repeat steps 1,…,3 until P_{fp} reaches a predefined value.

3. Experiment Setting

In this experiment, we experimentally evaluated the effect of four sampling methods (ROS, SMOTE, RUS, ONESS) applied to four commonly used fault-proneness models (linear discriminant analysis (LDA) [12][13], logistic regression analysis (LRA) [10], neural network (NN) [2] and classification tree (CT) [5]).

Since all four sampling methods can control P_{fp} (the percentage of fault-prone modules to all modules) in a sampled dataset, we also evaluated the prediction accuracy with different P_{fp} values. Note that P_{fp} should be 50% in terms of getting a balanced dataset; however, as described in section 2, there are two concerns with sampling (over sampling might cause over fitting to fake (added) cases, and, under sampling might eliminate useful cases), Thus, P_{fp} = 50 might not be the best choice.

So, first, as a pilot experiment, the best P_{fp} value for each sampling method was identified by using a fit dataset only (based on the cross-validation technique). Then, fault-proneness models were built after sampling methods were applied to a fit dataset with the best P_{fp} value. Finally, using a test dataset, the prediction performance of the fault-proneness models was compared to that of *naïve* fault-pronenesses models built without sampling methods.

Since all four sampling methods include some sort of randomness in their procedures, we repeated five times the above sampling, model building and performance evaluation and used the average performance values (recall, precision and F1-value).

Table 1. Collected metrics

	Metrics
m_1	Source lines of code
m_2	Commented lines per SLOC
m_3	The number of procedures per SLOC
m_4	The number of unique operators
m_5	The number of unique operands
m_6	Total number of operators per SLOC
m_7	Total number of operands per SLOC
m_8	Halstead volume
m_9	Halstead difficulty
m_{10}	Maximum of nest level
m_{11}	Cyclomatic number per SLOC
m_{12}	Nest level per SLOC
m_{13}	The number of jump nodes per SLOC
m_{14}	The number of external referred variables per SLOC
m_{15}	The number of inner calls per SLOC
m_{16}	The number of external module calls SLOC
m_{17}	Revision number
m_{18}	The number of days from the date each module is developed to the present
m_{19}	The number of days from the date each module is developed to the last release date

In this experiment, we used the data mining toolkit SPSS Clementine to build all four fault-proneness models. We applied step-wise variable selection to LDA and LRA.

Regarding the neural network model, five types of three-layer neural networks were constructed in the pilot experiment, each having one of the following numbers of learnings: 10,000; 20,000; 30,000; 50,000; 100,000. The best number identified in the pilot experiment was used in the main experiment. To determine the link weights, the error back propagation algorithm [14] is commonly used as a learning algorithm.

Also, as a classification tree model, we used the classification and regression trees (CART) algorithm [5] for model construction.

3.1. Dataset

The target is MIS (Management Information System) [4] software working on a mainframe machine. This software has been maintained for about twenty years and modified and expanded many times during that period. It was mainly written in two very old programming languages. The total size is about 1,000 KSLOC. Each language part consists of about 2000 modules (source files). Each module contains several procedures (subroutines).

In addition to fault data collected during maintenance, we measured 19 metrics for each module (Table 1). We used these 19 metrics as predictor variables. Fit datasets were built from data during three years prior to a certain release, and test datasets were built from data during three years after the release.

We separated modules into two sets A and B according to the programming language used. We label a fit dataset of language A, A_{fit} and language B, B_{fit}. Similarly, we label a test dataset of language A, A_{test} and language B, B_{test}. Original P_{fp} values of these datasets are shown in Table 2. The number of modules varies between fit datasets and test datasets due to the addition of new modules in the release.

3.2. Evaluation Criteria

We used three commonly used criteria, recall, precision and F1-value [3], to evaluate the prediction performance of built models. Recall is the ratio of correctly predicted fault-prone modules to actual fault-prone modules and precision is the ratio of actual fault-prone modules to the modules predicted as fault-prone. F1-value is a harmonic mean of recall and precision, formally defined as follows

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision}.$$

3.3. Evaluation Procedure

3.3.1. Finding appropriate percentage of fault-

Table 2. Statistics summary

	# of fault-prone modules	# of not fault-prone modules	% of fault-prone modules
A_{fit}	103	1815	5.67
A_{test}	42	1950	2.15
B_{fit}	210	1596	13.16
B_{test}	61	1815	3.36

prone modules (pilot experiment)

The goal of the pilot experiment is to identify the appropriate P_{ft} for each of 16 combinations of four sampling methods (ROS, SMOTE, RUS, ONESS) and four models (LDA, LRA, NN ,CT) by using a fit dataset. For each of 16 combinations, the following procedure was conducted.

Step 1. Dividing a fit dataset into two datasets

The fit data was divided into two datasets (denoted as *a* and *b*).

Step 2. Appling sampling methods

Apply a sampling method to dataset *a* to have P_{ft} = 20% (0.25 : 1.00), 33% (0.50 : 1:00), 50% (1.00: 1.00), 60% (1.50: 1.00) and 67% (2.00 : 1.00) in resultant datasets. Each new (resultant) dataset is referred to as a_1, …a_5; and, a dataset without sampling is reffered to as a_0.

Step 3. Building fault-proneness models

Build six fault-proneness models using datasets a_0, …a_5.

Step 4. Evaluation of the prediction performance

Evaluate the prediction performance (F1-value) of six fault-prone models by using dataset *b*.

Step 5. Finding appropriate P_{fp}

Select the best performing model to identify the best P_{fp}.

3.3.2. Evaluation of effect of sampling methods (main experiment)

In the main experiment, we built fault-proneness models after sampling methods were applied to a fit dataset with the best P_{fp} value and evaluated their prediction performance.

Step 1. Applying sampling methods to fit datasets

Apply sampling methods to fit datasets A_{fit} and B_{fit} with the best P_{ft} value.

Step 2. Building fault-proneness models

Build four fault-proneness models (LDA, LRA, NN ,CT) based on each resultant dataset of Step 1. Naïve models are also built based on A_{fit} and B_{fit} without applying sampling methods.

Step 3. Evaluation of the prediction performance

Evaluate the prediction performance (F1-value) of fault-prone models by using test datasets A_{test} and B_{test}.

4. Results and Discussion

4.1. Pilot Experiment

The prediction performance of fault-proneness models for different P_{fp} values are shown in Fig. 6. The vertical axis shows the F1-value and horizontal axis shows P_{fp}. The best P_{fp} values for datasets *A* and *B* are shown in Table 3 and Table 4.

As shown in Fig. 6, Table 3 and Table 4, it was revealed that the best P_{fp} values were not 50% in most models. Interestingly, this indicates that building a fully balanced fit dataset via sampling methods is not necessarily relevant. The following characteristics were found for each model.

LDA: The prediction performances got better as P_{fp} increased. The best P_{fp} values were 60%-67%. This indicates that sampling methods should be deeply applied so that minority cases (fault-prone modules) become majority cases in resultant fit datasets.

LRA: The result was totally different from LDA's. The best P_{fp} values were 20%-33%. This means that the sampling methods should be applied slightly.

NN: The result showed a somewhat similar tendency as LRA's. The best P_{fp} values were 20%-33% or without sampling. Notably, even slight sampling (P_{fp} = 20%) can cause performance degradation.

CT: There was no clear relationship seen between P_{ft} and the prediction performance.

4.2. Main Experiment

The result of the main experiment is shown in Tables 5 and 6. In the tables, "-" indicates a case where "no sampling" was the best in the pilot experiment; and, bold letters indicate performance improvements to naïve models.

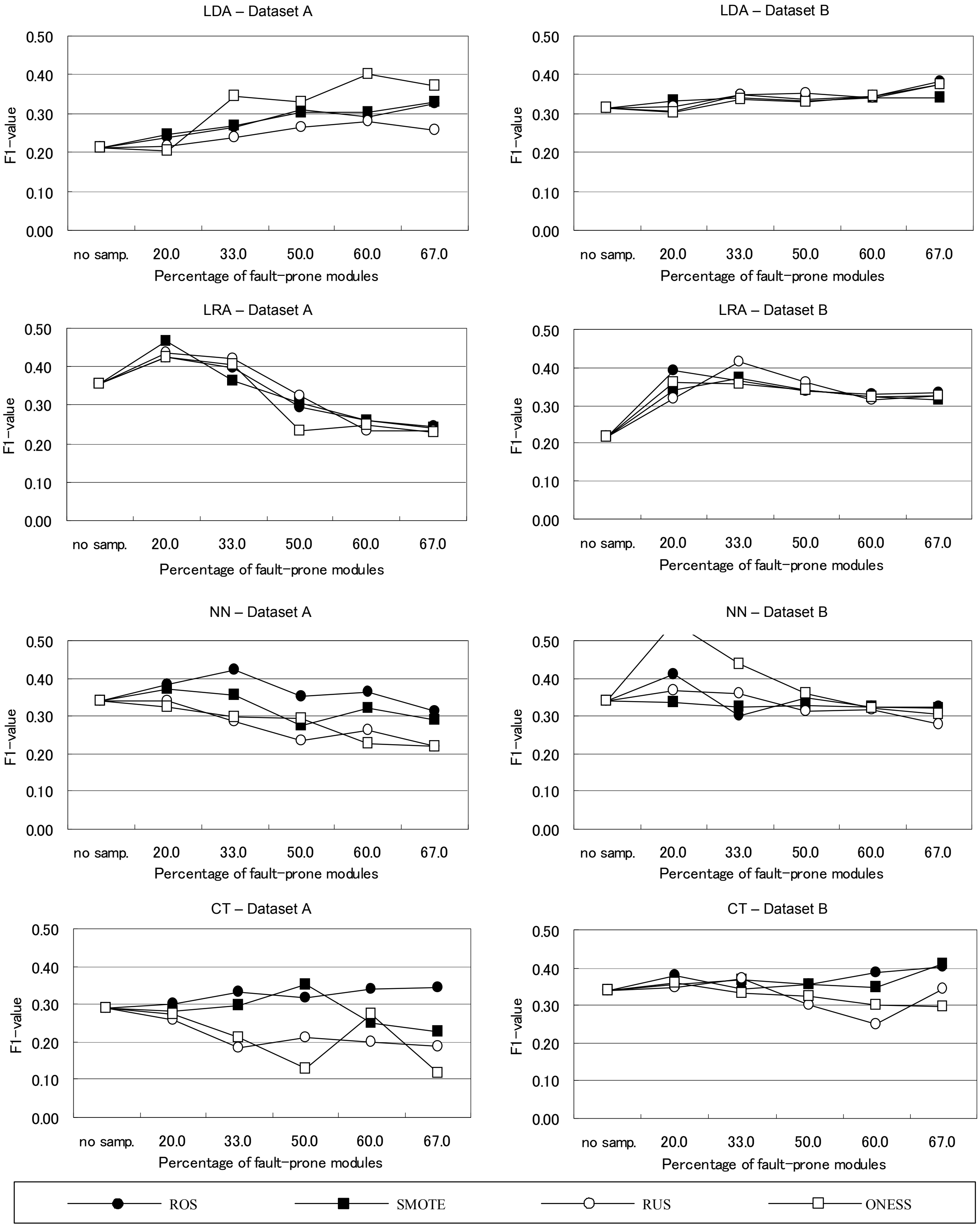

Figure 6. Prediction performance of fault-proneness models for different P_{fp} values

Table 3. Best P_{ft} values for dataset A

	LDA	LRA	NN	CT
ROS	67	20	33	67
SMOTE	67	20	20	50
RUS	60	20	20	No samp.
ONESS	60	20	No samp.	No samp.

Table 4. Best P_{ft} values for dataset B

	LDA	LRA	NN	CT
ROS	67	20	20	67
SMOTE	67	33	No samp.	67
RUS	67	33	20	33
ONESS	67	20	20	20

Comparison of fault-proneness models: The results showed that all four sampling methods improved the prediction performance of the LDA and LRA models.

On the other hand, the prediction performances of NN and CT were not improved in more than half of the cases. In addition, no sampling method could improve the performance for both datasets A and B in each model. This indicates sampling methods should not be applied to NN and CT models.

LRA showed the best prediction performance among the models for dataset *A*, and LDA was the best for dataset *B*. This result is consistent with Gray and Macdonells' result such that the best model among available models often depends on the dataset being used [2].

Comparison of sampling methods: The best performing sampling method was different among models and datasets. For example, for dataset A, a combination of SMOTE and LRA was the best (F1 value = 0.382) while for dataset *B*, a combination of RUS and LDA was the best (F1 value = 0.324).

With respect to LDA and LRA, the average prediction performance (F1-value) for each sampling method was nearly equal (ROS=0.276, SMOTE=0.287, RUS=0.293, ONESS=0.273). This suggests that we could use any of these four sampling methods for LDA and LRA. All in all, the improvements of F1-values were 0.078 at minimum, 0.224 at maximum and 0.121 at the mean.

5. Conclusion

In this paper, we experimentally evaluated the effects of four sampling methods (ROS, SMOTE, RUS, ONESS) applied to four fault-proneness models (LDA, LRA, NN, CT) by using two module sets of industry legacy software. Our major findings include the following:

Table 5. Prediction performance of models for dataset A

		LDA	LRA	NN	CT
Recall	No samp.	0.857	0.238	0.333	0.762
	ROS	0.752	**0.629**	**0.419**	0.648
	SMOTE	0.791	**0.595**	**0.443**	0.733
	RUS	0.833	**0.624**	**0.538**	-
	ONESS	0.795	**0.629**	-	-
Precision	No samp.	0.057	0.357	0.233	0.062
	ROS	**0.146**	0.266	0.142	**0.094**
	SMOTE	**0.137**	0.282	0.168	**0.080**
	RUS	**0.117**	0.257	0.136	-
	ONESS	**0.128**	0.269	-	-
F1-Value	No samp.	0.106	0.286	0.275	0.115
	ROS	**0.244**	**0.374**	0.212	**0.162**
	SMOTE	**0.233**	**0.382**	0.243	**0.143**
	RUS	**0.205**	**0.364**	0.217	-
	ONESS	**0.220**	**0.375**	-	-

Table 6. Prediction performance of models for dataset B

		LDA	LRA	NN	CT
Recall	No samp.	0.590	0.033	0.098	0.393
	ROS	0.420	**0.138**	0.069	0.325
	SMOTE	0.443	**0.220**	-	0.220
	RUS	0.482	**0.328**	**0.105**	**0.377**
	ONESS	0.505	**0.177**	**0.177**	**0.603**
Precision	No samp.	0.118	0.200	0.200	0.068
	ROS	**0.234**	**0.299**	0.099	0.061
	SMOTE	**0.249**	**0.210**	-	0.043
	RUS	**0.244**	**0.246**	0.098	0.052
	ONESS	**0.194**	**0.277**	0.144	0.060
F1-Value	No samp.	0.196	0.056	0.132	0.116
	ROS	**0.300**	**0.187**	0.081	0.101
	SMOTE	**0.319**	**0.215**	-	0.072
	RUS	**0.324**	**0.280**	0.100	0.090
	ONESS	**0.280**	**0.216**	**0.156**	0.108

- All four sampling methods improved the prediction performance of the LDA and LRA models while they could not always improve the performance of the NN and CT models. We recommend not using sampling methods for NN and CT models.
- The most appropriate sampling level varied among the models. For LDA models, the resultant fit dataset after applying a sampling method should have P_{ft} = 60%-67%. On the other hand, for LRA models, the resultant fit dataset should have P_{ft} = 20%-33%.
- With respect to the LDA and LRA models, the average prediction performance (F1-value) for each sampling method was nearly equal (ROS=0.276, SMOTE=0.287, RUS=0.293, ONESS=0.273). This suggests that we could use any of these four sampling methods for LDA and LRA. All in all, the improvements of F1-values were 0.078 at minimum, 0.224 at maximum and 0.121 at the mean.

The major limitation of this paper is that we used only two datasets. Our future work is to confirm our results using other datasets.

6. Acknowledgments

This work was partially supported by the EASE (Empirical Approach to Software Engineering) project, which is part of the Comprehensive Development of e-Society Foundation Software program of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] Chawla, N. V., Bowyer, K. W., Hall, L. O. and Kegelmeyer, W. P.: SMOTE: Synthetic Minority Over-sampling Technique, *Journal of Artificial Intelligence Research*, Vol. 16, pp. 321-357 (2002).

[2] Gray, A. R. and MacDonell, S. G.: Software Metrics Data Analysis. Exploring the Relative Performance of Some Commonly Used Modeling Techniques, *Empirical Software Engineering*, Vol. 4, No. 4, pp. 297-316 (1999).

[3] Herlocker, J. L., Konstan, J. A., Terveen, L. G. and Riedl, J. T.: Evaluating Collaborative Filtering Recommender Systems, *ACM Trans. Information Systems*, Vol. 22, No. 1, pp. 5-53 (2004).

[4] Jones, C.: Applied Software Measurement, Second Edition, p. 861, McGraw-Hill, New York (1996).

[5] Khoshgoftaar, T. M. and Allen, E. B.: Modeling Software Quality with Classification Trees, *Recent Advances in Reliability and Quality Engineering*, World Scientific, pp. 247-270, Singapore (1999).

[6] Khoshgoftaar, T. M., Gao, K. and Szabo, R. M.: An Application of Zero-inflated Poisson Regression for Software Fault Prediction, *Proc. 12th Int'l Symposium on Software Reliability Engineering (ISSRE'01)*, pp. 66-73, Hong Kong, China (2001).

[7] Khoshgoftaar, T. M., Yuan, X. and Allen, E. B.: Balancing Misclassification Rates in Classification-Tree Models of Software Quality, *Empirical Software Engineering*, Vol. 5, No. 4, pp. 313-330 (2000).

[8] Kubat, M. and Matwin, S.: Addressing the Curse of Imbalanced Training Sets: One-Sided Selection, *Proc. 14th Int'l Conf. on Machine Learning (ICML'97)*, pp. 179-186, Nashville, USA (1997).

[9] Li, P. L., Herbsleb, J., Shaw, M. and Robinson, B.: Experiences and Results from Initiating Field Defect Prediction and Product Test Prioritization Efforts at ABB Inc, *Proc. 28th Int'l Conf. on Software Engineering (ICSE'06)*, pp. 413-422, Shanghai, China (2006).

[10] Munson, J. C. and Khoshgoftaar, T. M.: The detection of fault-prone programs, *IEEE Trans. Software Engineering*, Vol. 18, No. 5, pp. 423-433 (1992).

[11] NASA IV&V Facility.: Metrics Data Program, http://mdp.ivv.nasa.gov/

[12] Ohlsson, N. and Alberg, H.: Predicting Fault-Prone Software Modules in Telephone Switches, *IEEE Trans. Software Engineering*, Vol. 22, No. 12, pp. 886-894 (1996).

[13] Pighin, M. and Zamolo, R.: A Predictive Metric Based on Discriminant Statistical Analysis, *Proc. 19th Int'l Conf. on Software Engineering (ICSE '97)*, pp. 262-270, Boston, USA (1997).

[14] Rumelhart, D. E., Hinton, G. E. and Williams, R. J.: Learning Representations by Back-propagating Errors, *Nature*, Vol. 323, pp. 533-536 (1986).

[15] Tomek, I.: Two Modifications of CNN, *IEEE Trans. Systems*, Man and Cybernetics, SMC.6, pp. 769-772 (1976).